UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Hash functions

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

http://netsec.unipr.it/veltri

Hash Function

- Also known as Message Digest
- It is a function that takes a variable-length input message and produces a fixed-length output

$$h = H(m)$$

  - **input message m of any size**
  - **output data *h* of fixed size**
  - **the output *h* is called message digest**
- The transformation H(m) is one-way
  - **it is not practical to figure out which input corresponds to a given output**
- Examples: MD5, SHA-1, SHA-2

Hash function properties

- Compression
  - **It reduces data size, “summarizing” the characteristics of the message**
    - input message of variable size
    - output of fixed size
      - is function of the entire input message
        - » allows the detection of possible modifications/errors
- Pseudorandomness
  - **the value of message digest should look “randomly generated”**
- Fast calculation (efficiency)
  - **given an input x, h(x) is easy (fast) to compute**
    - requires low processing resources

Hash function properties (cont.)

- Def: given a value *h*, if $H(x) \equiv h$ then *x* is called <u>preimage</u> of *h*
- Def: given a pair *x,y* with $x \neq y$, if $H(x) \equiv H(y)$, we have a <u>collision</u>
- One-way (or preimage resistance)
  - **for any output, it is computationally infeasible to find any input which hashes to that output**
    - given a value *h* (for which *m* is unknown), find *m'* such that $H(m') \equiv h$
- Weak collision resistance (or second-preimage resistance)
  - **it is computationally unfeasible to find any second input which has the same output as any specified input**
    - given *m*, find $m' \neq m$ such that $H(m') \equiv H(m)$
- (Strong) collision resistance
  - **it is computationally unfeasible to find any two distinct inputs *m*, *m*' which hash to the same output**
    - find *m* and *m*' such that $H(m') \equiv H(m)$

How many bits should the output have?

- How many bits should the output have in order to prevent someone from being able to find a collision?
- If the message digest has *n* bits, then it would take (expected value) $2^{n/2}$ messages chosen at random (Birthday Paradox)
  - **this is why message digest functions should have output of at least 160 or 256 bits (in place of just 128 as for symmetric cryptography)**
- However sometime it is not sufficient for an attacker to find out just two messages with the same hash
  - **in such case, a brute-force attack requires 2^n searches (mean value 2^{n-1})**
    - similarly to a brute force attack to a symmetric cipher

Preimage vs collision attack complexity (1/2)

- Preimage brute force attack complexity
  - **n = hash size**
  - **$N=2^n$ is the number of possible different hash values**
  - **P(k)= Pr{success with k tries} = Pr{success with k input messages}**
    - $P(1)= 1/N = 1 - (1-1/N)$
    - $P(2)= 1 - (1-1/N)^2$
    - $P(3)= 1 - (1-1/N)^3$
    - $P(k)= 1 - (1-1/N)^k$

    since $(1-x)^k \approx 1-kx$ when $x<<1$, then:
    - $P(k) \approx 1 - (1-k/N) = k/N$
  - **Look for the value of k such that $P(k) \geq 50\%$**
    - $P(k) \geq 1/2$
    - $k/N \geq 1/2$
    - $k \geq N/2 = 2^{n-1}$
- Same complexity of a brute force attack against a symmetric cipher secret key

Preimage vs collision attack complexity (2/2)

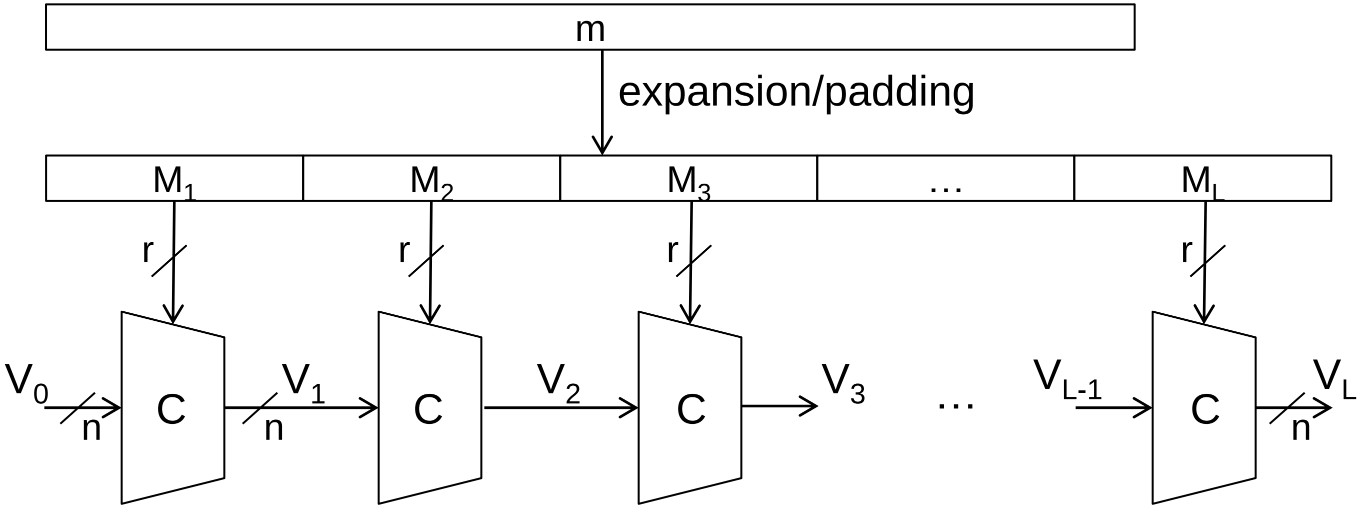
- Collision brute force attack complexity
  - **P(k)= Pr{success with k tries} = Pr{success with k input messages}**
    - $P(2)= 1/N = 1 - (N-1)/N$
    - $P(3)= 1 - (N-1)/N * (N-2)/N$
    - $P(k)= 1 - (N-1)/N * (N-2)/N * (N-3)/N * \ldots *(N-k+1)/N$
      $= 1 - (1-1/N) * (1-2/N) * (1-3/N) * \ldots *(1-(k+1)/N)$
  - **Since it is always $1-x \leq e^{-x}$, and if $x<<1$ then $1-x \approx e^{-x}$**
    - $P(k) \approx 1 - e^{-1/N} e^{-2/N} e^{-3/N} \ldots e^{-(k-1)/N} = 1 - \prod_{i=1,\ldots,k-1} e^{-i/N} = e^{-1/N \sum_{i=1,..k-1} i} =$
      $= 1 - e^{-k(k-1)/2N}$
  - **Look for the value of k such that $P(k) \geq 50\%$**
    - $P(k) > 1/2$
    - $e^{-k(k-1)/2N} < 1/2$
    - $k(k-1)/2N > \ln(2)$
    - $k^2 > 2N \ln(2)$
    - $k > \sqrt{N} \sqrt{(2 \ln 2)} \approx 1.18 * 2^{n/2}$
- If n bits are sufficient for resisting to preimage brute force attack, 2n bits are required to resists to a collision brute force attack
  - **two times the size of a symmetric key resistant to a brute force attack**

Birthday problem

- Collision on birthday
- Problem: How many people there should be in room in such a way the probability to have at least one birthday collision (two people have the same birthday) is greater than 50%?
- Solution:
  - **assuming randomly distributed birthdays, the probability that at least one person is born on a given day is approximately ≈ 23/365 = 0.063 (≈6%)**
  - **it is possible to calculate that with 23 people, the probability that at least two people have the same birthday is 0.507 (≈50%)**
  - **using the formula that we already obtained for a general collision attack:**
    - $k > 1.18 * \sqrt{365} = 22{,}54 \quad \Rightarrow \quad k \geq 23$

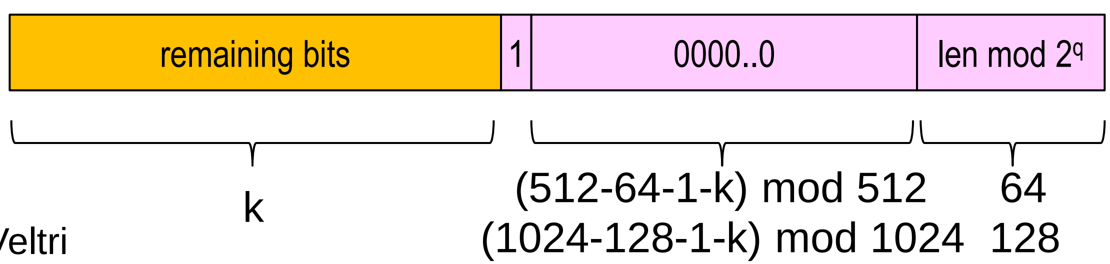
Structure of Hash functions

- Most hash functions *H* are designed as iterative processes which hash arbitrary length inputs *m* by processing successive fixed-size blocks of the input
  - Expand *m* to $(M_1,M_2,\ldots,M_L)$, with a total of $L \cdot r$ bits
  - For $i=1$ to L, compute $V_i = C(V_{i-1},M_i)$
  - Finally, set $H(m) = h = V_L$
- If appropriate padding is used and compression function *C* is collision-resistant, then the hash function is collision-resistant (Merkle-Damgard)



Padding

- Message processed in *r*-bit blocks
  - **input message needs to be padded in order to make the total length multiple of *r***
  - **message padding is not required to be invertible**
- Example of message padding (used by MD5, SHA)
  - **first bit is set to "1"**
  - **last *q* bits encode the length of the unpadded message mod 2^q**
  - **from *0* to up *r-1* bits set to "0" are added in the middle**
    - such that the total length of the padded message is multiple of *r*
  - **e.g.**
    - r=512, q=64
    - r=1024, q=128

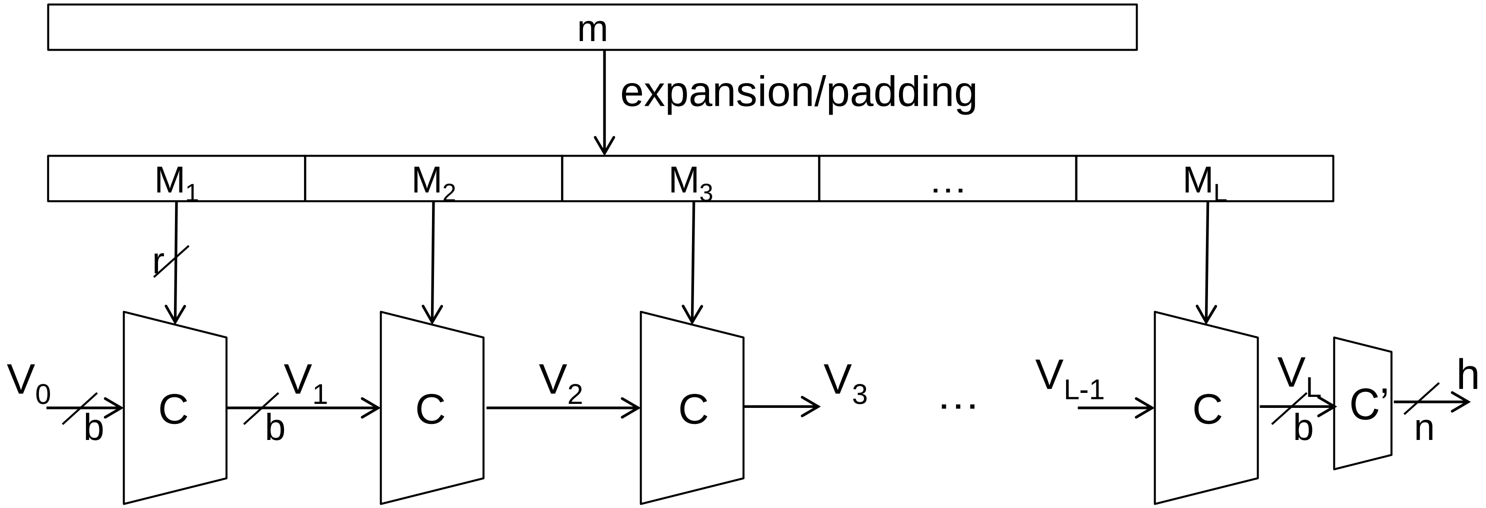


Structure of Hash functions (cont.)

- Possible attack to Merkle-Damgard hash functions:
  - Length extension attack
    - given $h=H(m)$, it is straightforward to compute m' and h', such that $h'=H(m||m')$, even for unknown m when the padding bits are known, (e.g. in case the padding bits are function of the m length known)
    - the attack is based on using h as an internal hash for computing h'

Structure of Hash functions (cont.)

- Wide-Pipe Hash (Stefan Lucks)
  - For $i=1$ to L, compute $V_i = C(V_{i-1},M_i)$, *b bits*
  - Finally, set $H(m) = h = C'(V_L)$, $n<b$ *bits*



Secure Hash Standard (SHS/SHA)

- Set of cryptographically secure hash algorithms specified by NIST as message digest functions
- The original specification of the algorithm was published in 1993 as the Secure Hash Standard, FIPS PUB 180, by NIST (SHA-0)
- Successively revised by the following standards
  - **SHA-1 (1995)**
    - like SHA-0, it produces a message digest that is 160 bits long
  - **SHA-224, SHA-256, SHA-384, SHA-512 (SHA-2, 2001)**
    - produce digests that are respectively 224, 256, 384, 512 bits long
  - **SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256 (2015)**
    - SHAKE128 and SHAKE256 produce variable length output
- Employed in several widely used security applications and protocols
  - **TLS/SSL, PGP, SSH, S/MIME, IPsec, etc.**

SHA standards

	Algorithm	Output size (bits)	Internal state (bits)	Block size (bits)	Word size (bits)	Rounds	Security $O(2^k)$
	SHA-1	160	160	512	32	80	<63(*)
SHA-2	SHA-224	224	256	512	32	64	112
SHA-2	SHA-256	256	256	512	32	64	128
SHA-2	SHA-384	384	512	1024	64	80	192
SHA-2	SHA-512	512	512	1024	64	80	256
SHA-3	SHA3-224	224	1600	1152	64	24	112
SHA-3	SHA3-256	256	1600	1088	64	24	128
SHA-3	SHA3-384	384	1600	832	64	24	192
SHA-3	SHA3-512	512	1600	576	64	24	256
SHA-3	SHAKE128	any	1600	1344	64	24	<128
SHA-3	SHAKE256	any	1600	1088	64	24	<256

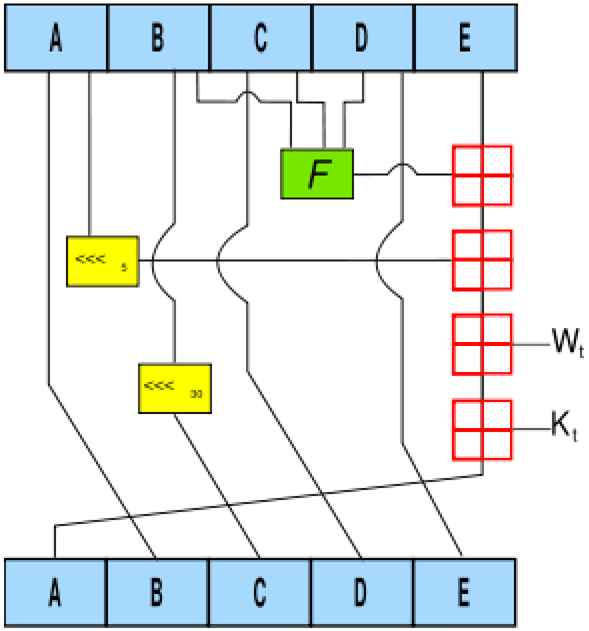
(*) SHA1 collision found (Feb, 2017)

SHA-1

- Secure Hash Standard 1 (SHA-1)
  - **published in 1995**
  - **differs from SHA-0 only by a single bitwise rotation in the message schedule of its compression function**
    - this was done to correct a flaw in the original algorithm which reduced its cryptographic security
- It produces a 160-bit (5 32bit-words) digest
- Based on principles similar to those used by MD5 message
  - **little slower than MD5 and little more secure**
- Operates in stages (as MD5)
  - **processes 512 bit blocks**
  - **uses the same padding mechanism of the MD5**

SHA-1 (cont.)

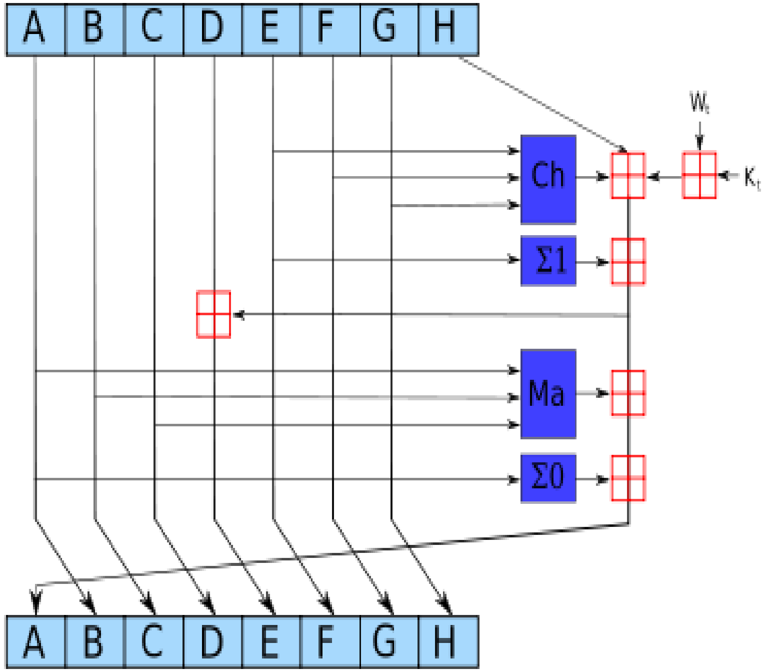
- Processing of one 512bit block of the input message:
  - **The 160bit state is view as 5x 32bit words**
    - A B C D E
  - **Each 512bit input block has 16 words**
    - $Y = X_0\ X_1\ ..\ X_{15}$
  - **Makes 80 rounds for each input block**
  - **In each round *t***
    - uses a word W_t derived by $X_0,X_1,..,X_{15}$
    - uses a different constant word K_t out of 4
    - uses a different function F_t out of 4
      - $Ch(x,y,z)=(x \wedge y)\oplus(!x \wedge z)$ $0 \le t \le 19$
      - $Parity(x,y,z)=x\oplus y\oplus z$ $20 \le t \le 39$
      - $Maj(x,y,z)=(x \wedge y)\oplus(x \wedge z)\oplus(y \wedge z)$ $40 \le t \le 59$
      - $Parity(x,y,z)=x\oplus y\oplus z$ $60 \le t \le 79$



$T = ROTL_5(A)+F_t(B,C,D)+E+K_t+W_t$
$E = D$
$D = C$
$C = ROTL_{30}(B)$
$B = A$
$A = T$
with additions mod 2^32 ⊞

SHA-2

- SHA-224, SHA-256, SHA-384, and SHA-512
- SHA-256 and SHA-512 are computed with 32- and 64-bit words, respectively
  - **use different shift amounts and additive constants**
  - **different number of rounds**
- SHA-224 and SHA-384 are simply truncated versions of the first two, computed with different initial values
- SHA-256 and SHA-512 perform 64 and 80 rounds, respectively



$T_1=H+\Sigma1(E)+Ch(E,F,G)+K_t+W_t$
$T_2=\Sigma0(A)+Maj(A,B,C)$
$H = G$
$G = F$
$E = D+T_1$
$D = C$
$C = B$
$B = A$
$A = T_1+T_2$

$Ch(x,y,z)=(x \wedge y)\oplus(!x \wedge z)$
$Maj(x,y,z)=(x \wedge y)\oplus(x \wedge z)\oplus(y \wedge z)$
$\Sigma0(X)=ROTR_{s1}(X)\oplus ROTR_{s2}(X)\oplus ROTR_{s3}(X)$
$\Sigma1(X)=ROTR_{s4}(X)\oplus ROTR_{s5}(X)\oplus ROTR_{s6}(X)$

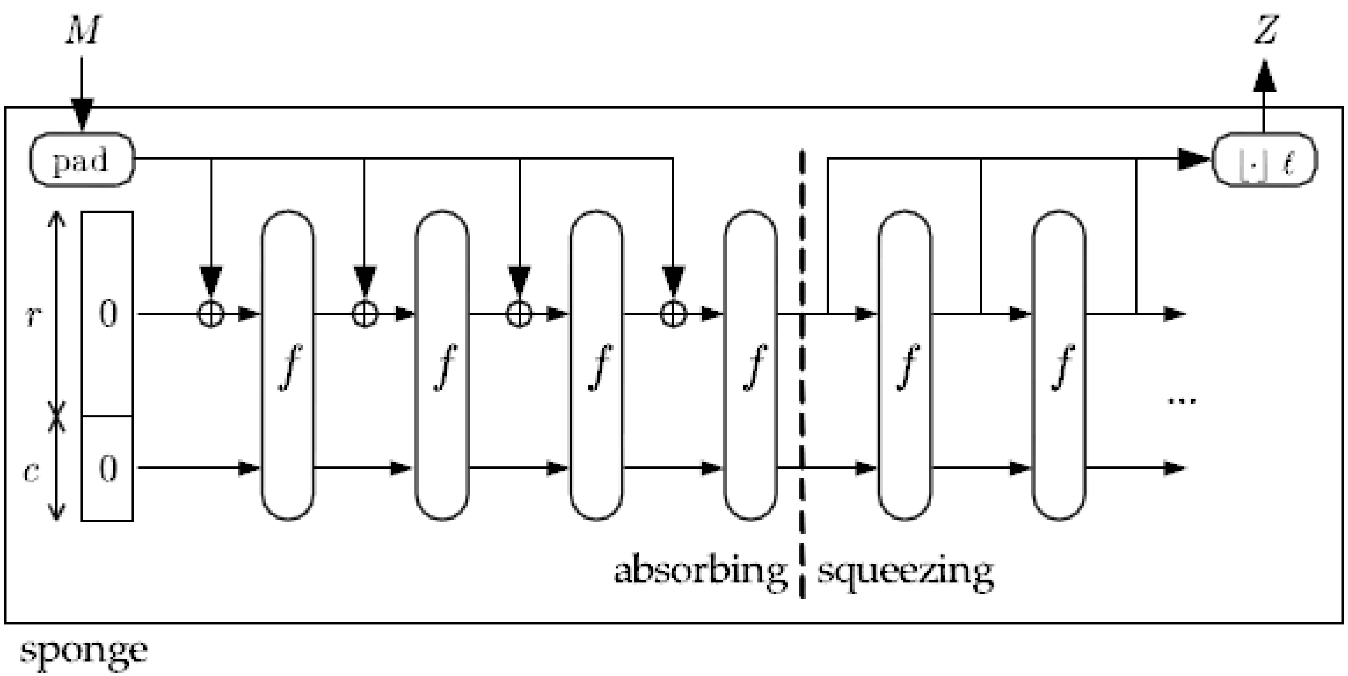
SHA-3

- SHA-1 has been attacked (collision attack)
  - **Complexity required for finding a collision is less then 2^{63}**
- SHA-2 security is not yet as well-established
  - **Not received as much scrutiny as SHA-1**
  - **Although no practical attacks have yet been reported, SHA-2 is algorithmically similar to SHA-1**
- SHA-3
  - **Chosen in 2012 after a public competition started by NIST in 2008**
    - similar to the development process for AES
    - NIST standard published in 2015
  - **Hash function formerly called Keccak**
  - **It supports the same hash lengths as SHA-2**
  - **4 cryptographic hash functions and 2 extendable-output functions**
  - **Its internal structure differs significantly from the rest of the SHA family. It is a *cryptographic sponge function***

Cryptographic Sponge Functions

- Generalize hash functions to more general functions whose output length is arbitrary
  - **variable-length input variable-length output function based on a fixed length transformation or permutation *f* operating on a fixed number *b* of bits (the width)**
    - f operates on a state of $b = r + c$ bits
      - the value r is called the bitrate and the value c the capacity
      - default values for Keccak are r = 576 bits, c = 1024 bits (b = 1600 bits)
  - **it processes blocks in two phases:**
    - the absorbing phase
      - the r-bit input message blocks are XORed into the first r bits of the state, interleaved with applications of the function f
    - the squeezing phase
      - the first r bits of the state are returned as output blocks Z_i, $i=1..k$, interleaved with applications of the function f
      - the number of iterations is determined by the requested number of bits ℓ
        - » $k=\lceil \ell / r \rceil$

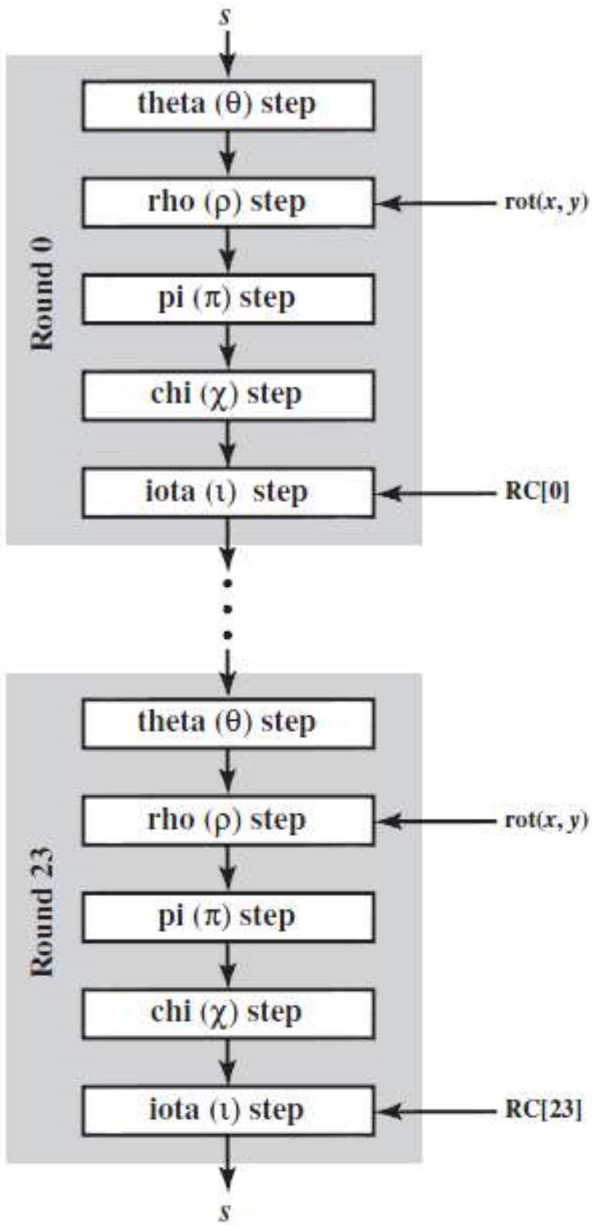
Cryptographic Sponge Functions (cont.)



- If the desired output length ℓ satisfies $\ell \leq b=r+c$, then at the completion of the absorbing phase, the first ℓ bits of the state are returned and the sponge construction terminates otherwise, the sponge construction enters the squeezing phase

SHA-3 (cont.)

- In SHA-3 that *f* takes as input a 1600-bit variable *s* consisting of *b*=*r*+*c* bits
- For internal processing within *f*, the input/internal state is organized as a 5 × 5 matrix of 64-bit words (referred as lanes) (1600 bits total)
  - *a*[x,y,z] is the bit array
  - *L*[*x*, *y*] is the 5x5 matrix
- The basic block permutation function KECCAK-*p* consists of 24 rounds of processing
  - **each round consists of five steps (functions) denoted by θ, ρ, π, χ, and ι**



What doing with a Hash

- Message fingerprint
  - **integrity check**
  - **maintaining a copy of a message digest of some data/program in place of the copy of the entire data**
- Password Hashing
  - **a system may know/store just the hash of a passwd**
- Digital signature
  - **signing the MD of a message instead of the entire message**
    - for efficiency (MDs are easier to compute than public-key algorithms)
- Entity authentication
  - **identification**

$r_A \longrightarrow$

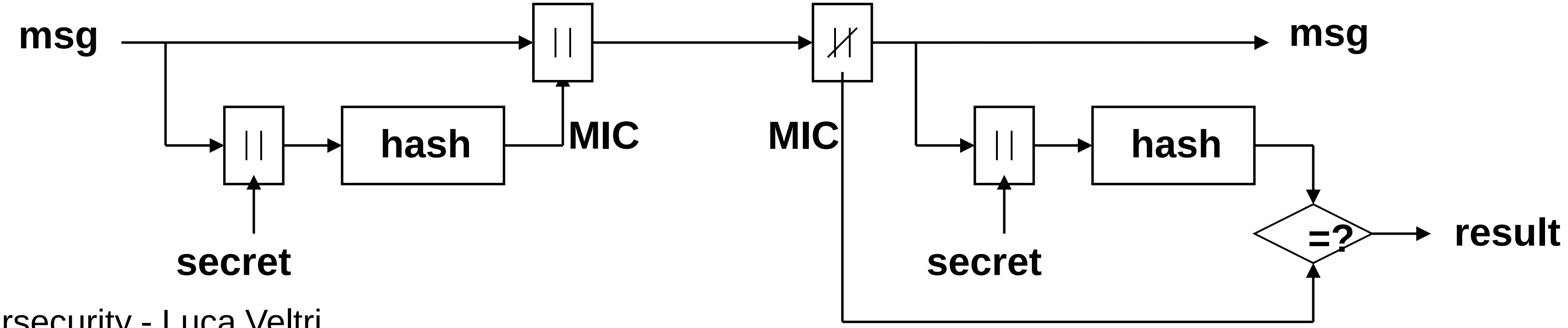
$\longleftarrow H(K_{AB}|r_A)$

$\longleftarrow r_B$

$H(K_{AB}|r_B) \longrightarrow$

What doing with a Hash (cont.)

- Message Authentication
  - ***H(m)* can be used as is a MIC for *m*, however:**
    - if not protected, can be modified by an intruder (anyone can compute *H(m)*)
    - cannot be used as cryptographic proof of the source
  - **possible solutions:**
    - encrypt *m* and *H(m)* with a secret key, or
    - compute the hash of both the message *m* and a secret
      - e.g. H(k||m)
      - the result is one-way function that takes two parameters: *k* and *m* (MAC function)

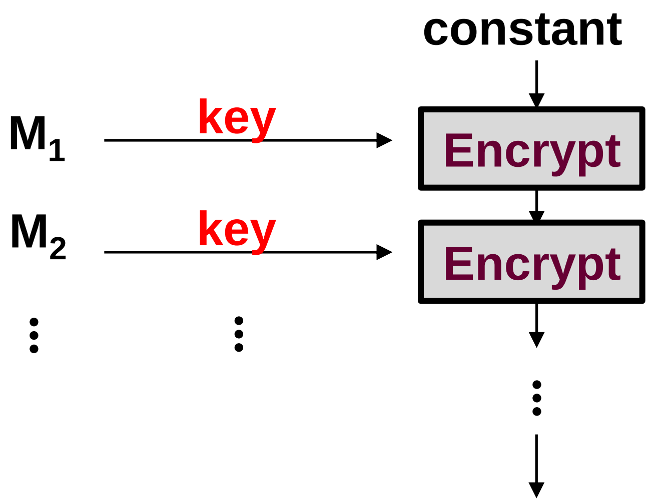


What doing with a Hash (cont.)

- Encryption
  - **An H function may be also used to build a cipher**
  - **one-time pad**
    - just as OFB (and CRT), generating a pseudorandom bit stream and encrypting the message just by a simple $\oplus$
    - the pseudorandom stream is generated starting from a hash of a secret
    - e.g. $O_1=H(K_{AB}|IV)$, $O_2=H(K_{AB}|O_1)$, .. , $O_i=H(K_{AB}|O_{i-1})$
    - same problems as OFB
  - **mixing in the plaintext**
    - as in CFB, the plaintext is mixed in the bit stream generation
    - $B_1=H(K_{AB}|IV)$, $B_2=H(K_{AB}|C_1)$, .. , $B_i=H(K_{AB}|C_{i-1})$
    - $C_1=M_1 \oplus B_1$, $C_2=M_2 \oplus B_2$, .. , $C_i=M_i \oplus B_i$

Using secret key algorithm for creating a Hash Function

- A hash function can be built by means of a block ciphers
  - **the message is padded ad divided in blocks**
    - $M_1, M_2, \ldots, M_n$
  - **each block is used to encrypt the output of the previous operation**
    - $H_i = E_{M_i} [H_{i-1}]$
    - $H_0=0$
  - **use final block as the hash value**



- Resulting hash can be too small (64-bit)
- Not very fast to compute

Example: Unix password hashing

- The orginal UNIX password hash "crypt function" uses DES to generate a hash of a password
  - **first convert the passwd (the message) into a “secret key”**
    - the 7bit ASCII codes of the first 8 chars form the 56bit key
  - **the key is used to encrypt the number 0 with a modified DES**
    - 25 DES passes are performed
    - the modified DES is used to prevent HW accelerators designed to DES to be used to reverse the passwd hash
    - the modified algorithm uses a 12-bit random number (salt)
  - **the salt and the final ciphertext are base64-encoded into a printable string stored in the password or shadow file**
- Other Unix/Linux password "crypt" functions have been added; currently they are:
  - **the original DES-based crypt function**
  - **hash-based functions (e.g. MD5-crypt function), where common hash function such as MD5 or SHA-1 are used**
    - such functions generally allow users to have any length password (> 8bytes), and do not limit the password to ASCII (7-bit) text

Example: Unix password hashing (cont.)

- The MD5-crypt function is really not a straight implementation of MD5
  - **first the password and salt are MD5 hashed together in a first digest**
  - **then 1000 iteration loops continuously remix the password, salt and intermediate digest values**
  - **the output of the last of these rounds is the resulting hash**
- A typical output of the stored password together with username, salt, and other information is:

alice:1BZftq3sP$xEeZmr2fGEnKjVAxzjQo68:12747:0:99999:7:::

  - **where 1 indicates the use of MD5-crypt, while BZftq3sP is the base-64 encoding of the salt and xEeZmr2fGEnKjVAxzjQo68 is the password hash**

Keyed Hash Functions

- Cryptographic one-way functions that create a small fixed-sized block depending on an input message *m* and a secret key K
  - $F_k(m) = F(k,m)$
- They condense a variable-length message *m* to a fixed-sized block
  - **often used as message authenticator**
  - **the result of the function and the function itself are usually referred to as Message Authentication Code (MAC)**
- MAC functions are similar to a Hash functions (one-way, collision resistant, etc.)
- The simplest way to build such a function could be to combine an Hash function with the secret key
  - **e.g.** $F(k,m) \equiv H(m||k)$
- Stronger functions can be designed, like HMAC (RFC 2104)
  - **see later**